PRELIMINARY DESIGN OF THE SYSTEMS IMPLEMENTATION LANGUAGE BLISS-360

Eugenia Mary Zavoyski

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

Preliminary Design of the
Systems Implementation Language BLISS-360

by

Eugenia Mary Zavoyski

Advisor: G. A. Kildall

June 1972

Preliminary Design of the Systems Implementation Language BLISS-360

by

Eugenia Mary Zavoyski
Major, United States Marine Corps Reserve
B.S., College Misericordia, 1958

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1972

ABSTRACT

The preliminary specification of a systems implementation language, called BLISS-360, is given. BLISS-360 is intended to be used for generation of software for the IBM S/360 computer, and is patterned after BLISS-10 which was originally designed for the PDP-10. The Backus-Naur Form description of BLISS-360 is given in a form acceptable to the Mixed-Strategy Precedence parser of the XPL Compiler Generator System. Thus, the XPL System can be used to aid future developmental efforts leading to a completed compiler for BLISS-360.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF DRAWINGS

ACKNOWLEDGEMENT

A word of thanks to Drs. Wulf and Habermann of the Computer Science Department of Carnegie-Mellon University for providing the documentation for BLISS-10.

I. INTRODUCTION

What is BLISS? Why is it important? These are some of the questions that must be answered in order to obtain an understanding of this new concept in programming languages. The following sections present details regarding its creation, design, and implementation in order to answer these questions.

A. BACKGROUND

This section provides the historical background of the language BLISS, the factors used as guidelines in its design, and a description of the language.

1. Historical Background

A research project on computer networks resulted in the acquisition of a PDP-10 computer at Carnegie-Mellon University in 1969. Part of this project consisted of producing a substantial number of large systems programs traditionally written in assembly language. Due to the inherent problems in the production and maintenance of such programs, members of the Computer Science Department staff (Wulf, Russell, and Habermann [Ref.1]) considered the use of a higher level language.

Because of the numerous types of languages available, the question of which to select was not an easy one to answer. For example, there are many languages in existence which are specifically designed

for systems programming such as BCPL [Ref. 2]. None, however, filled the criteria deemed important for this task. Therefore, a new language was created, called BLISS (Basic Language for the Implementation of Software Systems).

2. Design Objectives

BLISS was designed with several specific goals in mind. The goals covered such areas as the absolute requirements of systems programs, desirable features for systems programming, and language design.

The absolute requirements for all systems programs were considered to be efficiency, access to all hardware features of the machine, and minimal run-time support.

Those features deemed most important for systems programming include modularity, the ability to characterize a variety of data structures with diverse representations along with flexibility in the range of control structures.

In considering language design, stress was placed upon the requirements for a language which would be easily read, understood, and modified. The language would have to provide good documentation, be machine dependent, and be useful as a design tool.

It should be noted that access to all hardware features of the machine and machine independence are, clearly, incompatible.

Therefore, although many of its design objectives were achieved, BLISS is decidedly not machine independent. It was specifically designed for the PDP-10.

3. Description of BLISS

Described as a general purpose high-level language for writing production software for the PDP-10, BLISS is strictly a modular language. Its primary program element is a module which can be separately compiled and later linked with other object modules to form a complete program. The module consists of one or more executable expressions, each of which computes a value. Thus, BLISS can be characterized as an "expression language" [Ref. 1].

There are many similarities between BLISS and ALGOL or PL/I. For example, BLISS is a block structured language with variables which obey the usual scope rules. Its expressions are similar in format. It provides the usual conditional and looping constructs, operator hierarchy, and potentially recursive procedures.

Storage is dynamically allocated at block entry and deallocated at block exit for "local" and "register" variables; for "own" and "global" variables storage is statically allocated in reserved areas of core and remains allocated during execution of the entire program. There is, however, no "type" associated with any storage segment (a fixed and finite number of 36-bit words), as in ALGOL.

Each word or any contiguous set of bits in a word is called a "field" and, as such, may be named. The value of a name, however, is a "pointer" to, or the address of, that field, as opposed to the familiar "contents of" interpretation.

Since there are no built-in data structures in BLISS, access to the "programmer-defined" data structures is through "programmer-declared" accessing algorithms.

The familiar "go to" is replaced by additional control expressions to improve readability and structuring of programs. These expressions, modified in BLISS-11 (PDP-11 version of BLISS-10) [Ref. 3], allow for exiting various control environments such as blocks, compound expressions, and looping expressions.

A time-saving, but space-consuming [Ref. 4] feature is the activation of the body of a function or routine as a coroutine and/or asynchronous process.

Highly machine dependent features include the ability to insert machine language instructions inline with the other expressions.

4. Importance of the Implementation of BLISS

BLISS adds some interesting new concepts to the field of programming languages. In addition, the ease with which it can be read and understood facilitates redesign or restructuring.

BLISS has been proven to be useful as a design tool. For example, it has been used to write such systems as a WATFOR-like fast FORTRAN compiler and a SIMULA-like event-based simulation system [Ref. 1].

It seems apparent that if BLISS were implemented on other computer systems, programming tasks would be accomplished much more efficiently with lower overhead costs in time and effort.

B. OBJECTIVES

The overall goal of this project is to design and implement a BLISS compiler for the IBM S/360. The first step, however, in reaching this goal is the redesign of the language. Thus, the objective of this thesis is to tailor or redesign the BLISS language for implementation on the IBM S/360 without greatly compromising the original design. (Reference 5 presents further work toward the achievement of the overall goal.)

II. DESIGN APPROACH

In redesigning a machine dependent language for implementation on another computer, it is imperative to understand the architecture of both computers in order to determine those aspects of the language which will have to be modified. In addition, it is important to be familiar with the particular technique which will be used to implement the language. The following sections discuss the approach taken in the implementation of the BLISS-360 language.

A. EVALUATION OF MACHINE DEPENDENCIES

The purpose of this section is to present a comparison of the pertinent features of the PDP-10 and the IBM S/360 and to discuss the effect of their differences on BLISS. (Detailed information concerning the architecture of the PDP-10 and the IBM S/360 is given in Refs. 6 and 7, respectively.)

1. Characteristics of the Computers

This section provides a comparison of the hardware capabilities of the PDP-10 and IBM S/360 computers.

The following are the significant features of the PDP-10. There are sixteen general registers which can function as accumulators or index registers. In addition, these registers can be referenced as normal memory locations 0_8 through 17_8. The basic addressable unit is

the word which consists of 36 bits. The following data types are available: fixed-point numbers, floating-point numbers, logical operands, and the byte - any contiguous set of bits within a word.

The instruction set on the PDP-10 consists of fixed and floating-point instructions as well as a set of five byte manipulation instructions. There are two primary types of instruction formats - the basic format and the in-out format. In addition, the PDP-10 provides another format called the Byte Pointer as the pointer to, or location of, a byte to be manipulated. These formats are shown in Figure 1. Note that "address type" refers to the type of addressing to be performed, i.e., direct or indirect.

Following are a number of significant features of the IBM S/360. There are sixteen general registers which can be used as accumulators for fixed-point arithmetic, relocation registers, index registers, and for logical operations. In addition, there are four floating-point registers for floating-point operations. The basic addressable unit is the 8-bit byte. Other addressable units include the halfword, word, and the doubleword. It must be noted that since the fixed-length fields, i.e., the halfword, word, and doubleword, can be addressed, they must be located on an integral boundary in main storage. A halfword, for example, must have an address which is a multiple of two. Variable-length fields, however, may start at any byte address.

PDP-10 INSTRUCTION FORMATS

BASIC INSTRUCTION FORMAT

Instruction Code	Accumulator Address	Address Type	Index Register Address	Memory Address
0 8	9 12	13	14 17	18 35

IN-OUT INSTRUCTION FORMAT

7_8	Device Code	Operation	Address Type	Index Register Address	Memory Address
0 2	3 9	10 12	13	14 17	18 35

BYTE POINTER

Position	Size		Address Type	Index Register Address	Memory Address
0 5	6 11	12	13	14 17	18 35

Figure 1

The data types which are available on the IBM S/360 include the character (byte), fixed-point numbers, floating-poing numbers (single and double precision), decimal numbers, and logical operands.

The instruction set consists of fixed and floating-point instructions and an extensive set of byte manipulation instructions. There are five basic instruction formats including register-to-register operation (RR format), register-and-indexed storage operation (RX format), register-and-storage operation (RS format), storage-and-immediate operand operation (SI format), and storage-to-storage operation (SS format). These formats are shown in Figure 2.

2. Effects of the Machine Differences on BLISS

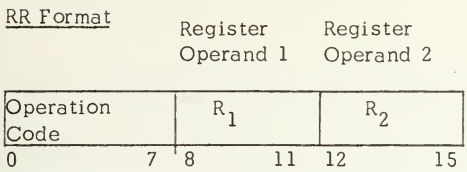
Examination of the differences between the PDP-10 and the IBM S/360 computers indicates that several modifications are required to the BLISS language in order to incorporate such features as machine language declarations, "contents of" operators, and pointer parameters. In addition, in order to take advantage of the architecture of the IBM S/360, additional language constructs for the arithmetic expressions and modifications to the allocation declarations are required.

B. TECHNIQUE USED IN THE IMPLEMENTATION OF THE LANGUAGE

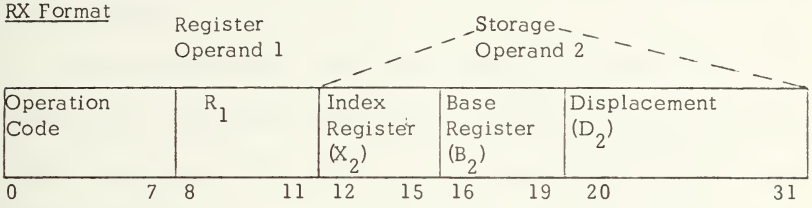
The technique to be used for writing a compiler is of primary concern in the implementation of a language. For example, the compiler could be initially written in assembly language for a small subset of

IBM S/360 INSTRUCTION FORMATS

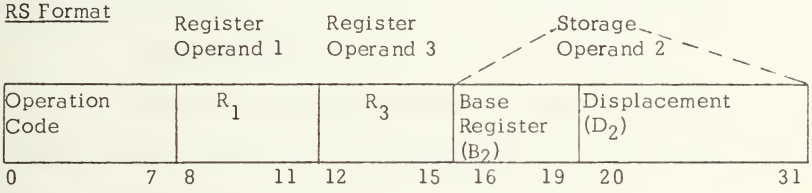
RR Format



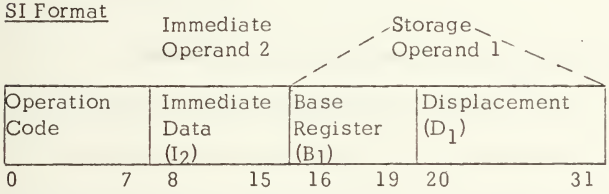
RX Format



RS Format



SI Format



SS Format

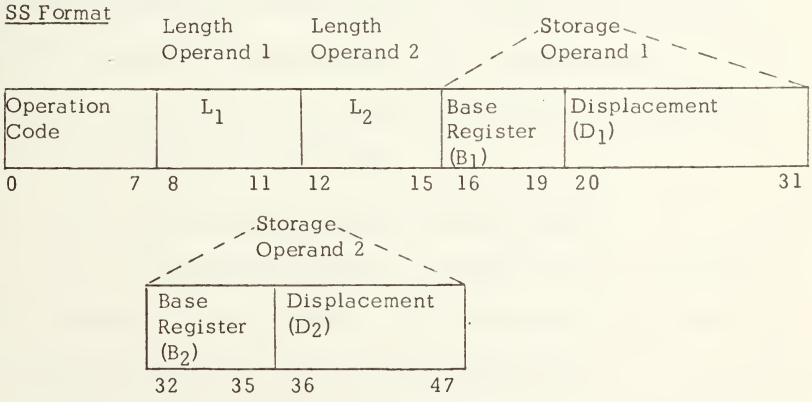


Figure 2

BLISS. Subsequent steps would include rewriting the compiler in the language itself with successive extensions implementing desired features of the language.

The technique used in the initial stages of the implementation of BLISS-360 is a Translator Writing System (TWS). The following sections provide an overview of the TWS used - McKeeman's XPL Compiler Generator System [Ref. 8].

1. The XPL System

The XPL System was specifically designed as a tool for writing compilers for the IBM S/360. The components of the XPL System are discussed in this section.

The XPL System consists of three principal components which are expressed in the XPL language (a dialect of PL/I). These components are the Analyzer, which analyzes a source language syntactically defined in BNF and generates parsing tables; the Skeleton, which performs a syntax analysis of a program written in the source language; and XCOM, which translates XPL to S/360 machine code. Only the details of the Analyzer component are given below.

2. The Analyzer Component of the XPL System

The Analyzer component is concerned with the analysis of the grammar of a source language. This section provides a general discussion of the method it uses to perform this function.

The Analyzer reads a phrase-structure grammar expressed in Backus-Naur Form (BNF) and determines its acceptability to the Mixed-Strategy Precedence (MSP) parsing algorithm. The method used in determining the acceptability of the grammar follows. Initially, the grammar is checked for certain forms of ambiguities, i.e., it is checked to determine whether there exists a sentence in the grammar for which there is no unique canonical parse. The stacking decision function (C1) is then computed. The purpose of this function is to decide whether the next input symbol should be placed on the parse stack or whether the symbols in the parse stack should be reduced. The MSP algorithm uses a C1 function of degree (1,1) first, i.e., it looks at one symbol on top of the stack and one in the input text. In cases where the (1,1) predicate is undefined (no decision can be made), it reverts to degree (2,1), i.e., two symbols on top of the stack are examined. When all stacking decision conflicts are resolved, the production selection function (C2) is computed. This function checks the context for equal or embedded right parts and determines the correct production to be selected. The MSP algorithm uses a C2 function of degree (1,1), i.e., the correct production must be chosen by using no more than one symbol below the production in the parse stack along with the next symbol in the input text.

After the grammar is determined to be acceptable to the MSP algorithm, the Analyzer produces tables, such as the stacking tables and the context checking tables, to be used for the analysis algorithm of the Skeleton component.

3. The Use of Analyzer in the Implementation of BLISS-360

The Analyzer will accept grammars consisting of a maximum of 255 productions. The BLISS-360 language currently consists of 238 productions; therefore, it can be accepted by Analyzer. It should be noted, however, that in the event more than 17 additional constructs are added to the language, the data-packing procedures in Analyzer will have to be re-written. The alternative solution to this would be the segmentation of the grammar allowing several passes in the syntax analysis. This form of grammar analysis was considered in the initial design stages and determined to be not feasible because of the extent of self-embedding in the language.

Although BLISS-360 can be accepted by Analyzer by virtue of its size, Analyzer produces a large number of tables for the language. Therefore, it was necessary to increase the limits built into Analyzer on the size of several tables.

III. RESULTS

Upon evaluation of the design considerations presented in the previous section, several modifications were required to BLISS-10 in order to implement the language on the IBM S/360. This section discusses these modifications and provides a general description of the resultant language, BLISS-360. In addition, the features to be incorporated in the future version of BLISS-360 will be described.

A. MODIFICATIONS TO THE ANALYZER COMPONENT OF THE XPL SYSTEM

BLISS is a large language full of self-embedded structures. Therefore, in order to make the language acceptable to the Analyzer component of the XPL System, several modifications were made to Analyzer. These changes are listed in Table I. It should be noted that the value of those parameters not reflected in the output listing from Analyzer are left blank in the table.

As a result of these changes, the Analyzer program requires a region of 250 K bytes in which to perform the grammar analysis.

B. DESCRIPTION OF THE LANGUAGE BLISS-360

BLISS 360 is most easily described by referring to similar constructions in BLISS-10 since BLISS-360 is intended to perform the same tasks for the IBM S/360 as BLISS-10 does for the PDP-10.

TABLE I

MODIFICATIONS TO ANALYZER

Parameter	Use	Original Version of Analyzer	Modified Version of Analyzer	Size Required by BLISS-360
Depth	Level of recursion	255	1023	816
Maxnfll	Maximum number of contexts for each rightmost non-terminal in all the canonical sentential forms	5000	30000	25136
Maxntrip	Maximum number of C1 triples	1000	4000	553
Stacklimit	-	200	500	-
Textlimit	Maximum number of symbols in the input string	255	511	-
Maxtrouble	Maximum number of contexts for which C1 and C2 are undefined	50	100	-

As is evident from the complete BNF description of the current version of BLISS-360 in Appendix A, most of the richness of BLISS-10 has been retained. However, there are fundamental differences between the two languages. It is the purpose of this section to discuss the differences between BLISS-10 and the current version of BLISS-360.

In order to more fully understand BLISS-360 and the following discussion, it is recommended that the reader familiarize himself with BLISS-10 [Ref. 9].

BLISS-10 provides the ability to control the actions of the compiler with respect to the program through the use of the "parameters" field in the module definition and in the "switches" declaration. This capability will be provided, but is not reflected in the syntax.

In BLISS-10, the reserved word SEMICOLON indicates that side effects could occur in the previous expression; thus, any code optimization must take this into account. SEMICOLON is, however, syntactically identical to the symbol ";" and is not included in the BLISS-360 description.

The literal construct has been modified to include "identifier" which replaces "name" in all relevant constructs. The symbol "identifier" in BLISS-360 is syntactically and semantically identical to "name" in BLISS-10. It should be noted that the symbols comprising the "literal" construct are used as terminal symbols to be recognized by the Scan Routine in the Skeleton by their use. Another symbol, "string," is used

similarly. Note that the "plit" construct is not included in the current version of BLISS-360.

The symbol "number" in BLISS-360 differs both in syntax and semantics from "number" in BLISS-10. In BLISS-360, "number" is syntactically defined by the following BNF notation:

```
<Number> : : = <Fixed point> | <Floating point> | <Packed decimal>

<Fixed point> : : = <Fullword> | <Halfword>

<Fullword> : : = <Decimal number> | <Hexadecimal number>

<Decimal number> : : = <Digit> | <Decimal number> <Digit>

<Hexadecimal number> : : = # <Hex digit> |
                           <Hexadecimal number> <Hex digit>

<Halfword> : : = <Decimal number> H

<Floating point> : : = <Long real> | <Short real>

<Long real> : : = <Short real> L

<Short real> : : = <Decimal number> . |
                   <Short real> <Decimal number> |
                   <Short real> <Exponent>

<Exponent> : : = E + <Decimal number> | E - <Decimal number>

<Packed decimal> : : = <Decimal number> P

<Digit> : : = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Hex digit> : : = <Digit> | A | B | C | D | E | F
```

Several examples are provided to illustrate the various types of numbers described:

```
123H    #1D    787.8E+9L    59.E-10    4096P
```

The number 123H represents the integer constant 123 requiring one halfword of storage, while the number #1D represents the hexadecimal equivalent of the integer constant 29 requiring one fullword of storage. The number 787.8E+9L represents the long real (double precision) floating-point number 787.8×10^{9}; the number 59.E-10 represents the short real (single precision) floating-point number 59.0×10^{-10}, and 4096P represents the integer constant 4096 in packed decimal format requiring three bytes of storage.

Because the PDP-10 provides the capability of addressing any contiguous field in a memory word, BLISS-10 allows left and right justification of character strings, denoted by single and double quotes, respectively. The strings are limited in size to the number of characters which can be represented in a word. The size is dependent upon the type of code used to specify the characters. (The PDP-10 computer uses 7-bit ASCII, 6-bit ASCII (SIXBIT), and RADIX-50 codes.)

The byte, the smallest addressable unit on the IBM S/360, represents one character in the character string. Consequently, BLISS-360 does not allow for left and right justification in strings and only allows single quotes to be used to enclose the string. The number of characters in the string is not limited to the word size. It should be noted that

although hardware is available to process 8-bit ASCII codes, the EBCDIC coding scheme is primarily used on the IBM S/360. Therefore, only EBCDIC strings are allowed in BLISS-360.

BLISS-10 allows the inclusion of an "escape to control" character, represented by the symbol "?" in a string. For example, "?M" is interpreted as "control-M," or carriage-return. Since this control character only has meaning for the PDP-10, the symbol "?" in a BLISS-360 string will have no special meaning. A double quote within a string is taken as a single quote.

Another feature of BLISS-10 utilizing the hardware capabilities of the PDP-10 is the "pointer parameter" construct. A"pointer parameter" is the address of a field of a contiguous number of bits within a word (byte). It consists of five parameters on the PDP-10 which are encoded into a single word - the Byte Pointer shown in Figure 1. The usefulness of the "pointer parameter" is readily apparent; it allows access to, and manipulation of, a particular byte within a word. Since the IBM S/360 does not have the equivalent of the Byte Pointer, it will be necessary to implement the "pointer parameter" through subroutine calls or sequences of shift and mask operations. This construct, however, is not included in the present BLISS-360 grammar.

There are three "contents of" operators in BLISS-10: the dot, the "@" operator, and the "\" operator. When these operators are used with an identifier, they allow direct access to, and control over, fields

within words and pointers to fields stored in memory. Only the dot, representing the value of the identifier, is defined in BLISS-360. The size of the value is dependent upon the boundary alignment of the identifier. For example, if an identifier is aligned on a halfword boundary, then the value obtained will be 16-bits in length. Since the value of the entire address is fetched, the "@" and the " \" operators would be redundant in BLISS-360.

Several formats are provided in BLISS-360 to resolve the "dangling else" problem of the conditional expressions in BLISS-10. Similar to the format proposed by McKeeman in Ref. 8, the BLISS-360 constructs always associate the ELSE with the closest unmatched IF-THEN.

The BLISS-10 increment and decrement expressions which are similar to the ALGOL FOR statement [Ref. 10] allow for the omission of any of the phrases "FROM E_1", "TO E_2", or "BY E_3." BLISS-360 only provides for the omission of the "BY E_3" phrase. The default value "BY 1" is supplied in the event this phrase is omitted.

The familiar "go to" expression is not allowed in the BLISS-10 language. Instead, BLISS-10 provides an "escape" expression as the form for exiting control environments such as the block. The "escape" expression consists of a wide variety of exit expressions such as EXITBLOCK. Each exit expression causes control to leave one or more of the specific control environments in which the exit expression is

embedded and pass onto the next expression following the outermost environment exited.

In order to reduce the number of exit expressions, BLISS-11 was designed with a "leave" expression replacing all exit expressions [Ref. 3]. The "leave" expression functions the same way as the "escape" expressions. The difference between the two, however, is that the control environment to be exited is not specified by the particular exit expression used; instead, it is identified by the name which points to the location in the program in which it occurs. In other words, the control environment is labelled.

The BLISS-360 language modified the syntax of BLISS-10 for the "escape" expression allowing only the EXITLOOP and RETURN expressions from BLISS-10 and adding the "leave" expression from BLISS-11. The syntax for the BLISS-360 "escape" expression follows:

< Escape expression > : : = < Environment escape > |
< Procedure escape >

< Environment escape > : : = < Loop escape > |
< Control environment escape >

< Loop escape > : : < Exit > | < Exit expression >

< Exit > : : = EXITLOOP

< Exit expression > : : = EXITLOOP < Exit end >

< Exit end > : : = < Braced E > | < Escape value > |
< Braced E > < Escape value >

< Braced E > : : = < < E > >

< Escape value > : : = < With clause >

< With clause > : : = WITH < Unlabelled expr >

< Control environment escape > : : = < Labelled leave > |

< Labelled leave > < Escape value >

< Labelled leave > : : = LEAVE < Identifier >

< Procedure escape > : : = < Return > | < Return expression >

< Return > : : = RETURN

< Return expression > : : = RETURN < Escape value >

Although the syntax of the EXITLOOP and RETURN expressions has been modified, the semantics remain the same.

The "leave" expression terminates the labelled control environment in which the "leave" expression is embedded. The value of the "leave" expression is that obtained by evaluating the "escape value." If the "escape value" is omitted, the value of the "leave" expression is zero.

An example of the use of the "leave" expression is provided:

```
WHILE .A NEQ .B DO
    LABELIT: IF .B GTR 59.5L THEN LEAVE LABELIT WITH .B
                ELSE A : = .A + .B/2.L
```

In conjunction with the "leave" expression, the BLISS-360 language provides a "label" declaration and a modified syntactical description of "expression." The "label" declaration is syntactically defined as:

< Label declaration > : : = < Label head > < Identifier >

< Label head > : : = LABEL | < Label head > < Identifier > ,

The "label" declaration is similar, in only one respect, to the forward function declaration in that it names the identifiers to be used as labels later in the block in which the declaration occurs.

Example:

```
LABEL LABELIT, LABEL1, LABEL2
```

The "expression" in BLISS-360 allows for both labelled and unlabelled expressions. It is defined by the following BNF:

< E > : : = < Labelled expr > | < Unlabelled expr >

< Labelled expr > : : = < Label > : < Unlabelled expr >

It should be noted that "label" must have been previously declared by a "label" declaration. The unlabelled expression is exactly equivalent to "expression" as defined in BLISS-10.

In order to avoid conflicts in parsing BLISS-360 programs, several reserved words in the allocation declaration were changed. The changes are listed in Table II.

It should be noted that since the general-purpose and floating-point registers on the IBM S/360 cannot be referenced as memory locations, storage segments for the REGALLOC, FPREGALLOC, REGISTER, and FPREGISTER variables will be the same as for the LOCAL variables in BLISS-10.

BLISS-360 incorporated the BLISS-10 feature of allowing the insertion of machine language instructions into the BLISS program. However, because of the variety of instruction formats used by the IBM S/360, the formats of the machine language declaration and the inline code are modified.

The form to be followed for the machine language declaration in BLISS-360 will resemble the "function" declaration of PL360 [Ref. 11]. An example of the BLISS-360 machine language declaration is:

```
MACHOP MVC = (5, #D200)
```

where MVC is the mnemonic name of the operation code, the number "5" defines the format of the instruction, and the hexadecimal number "D200" defines the first two bytes of the instruction. Note that "D2" is the hexadecimal equivalent of the operation code MVC.

The format to be followed for the inline coding of machine language instructions is the same as the format used for the "function" statements in PL360.

An example of inline code in BLISS-360 follows:

```
MVC (10, Field1, Field2)
```

where the number "10" specifies the length of the receiving field, "Field1" provides the location of the receiving field, and "Field2" provides the location of the sending field.

TABLE II

BLISS-360 CHANGES TO RESERVED WORDS
IN THE ALLOCATION DECLARATION

Reserved Word in BLISS-10	Declaration in which used	Reserved Word in BLISS-360
REGISTER	Register allocation declaration	REGALLOC
EXTERNAL	External allocation declaration	EXTALLOC
-	Floating-point register allocation declaration	FPREGALLOC
REGISTER	Alternate register allocation declaration	REGISTER
-	Alternate floating-point register allocation declaration	FPREGISTER

Several symbols used in BLISS-10 are not available for use in BLISS-360. Table III contains a listing of the symbols replaced in BLISS-360.

Finally, note that BLISS-360 programs must contain the reserved word "EOF" signalling the end of the program.

TABLE III

SYMBOLS REPLACED IN BLISS-360

Symbol used in BLISS-10	Equivalent symbol used in BLISS-360	Use
[	<	Delimiter for "structure access" and "structure" declaration.
]	>	Same as above.
!	?	Comment delimiter.
←	: =	Assignment operator.
↑	|	Shift operator.

C. FUTURE DEVELOPMENTS IN THE DESIGN OF BLISS-360

To complete the design phase of the BLISS-360 language, several additions and modifications will be required. The changes discussed in this section will allow BLISS-360 to fully utilize the hardware capabilities of the IBM S/360. In addition, they will provide BLISS-360 with all the richness of BLISS-10.

In order to provide flexibility in the types of arithmetic operations to be performed in the BLISS-360 program without resorting to inline coding of machine language instructions, the "P5" and "P6" constructs will be modified as follows:

```
<P6>  ::= <P5> | <Neg> <P5> | <P6> <+> <P5> |
          <P6> <-> <P5>
<Neg> ::= NEG | FNEG | PNEG
<+>   ::= + | ADH | ADP | ADL | ADS
<->   ::= - | SUBH | SUBP | SUBL | SUBS
<P5>  ::= <P4> | <P5> <*> <P4> | <P5> </> <P4> |
          <P5> MOD <P4>
<*>   ::= * | MULH | MULP | MULL | MULS
</>   ::= / | DIVP | DIVL | DIVS
```

The "NEG" construct provides fixed-point, floating-point, and packed decimal "negative" operations. The other types of arithmetic operations are defined by the symbol or the final letter of the reserved word as follows: fixed-point halfword is specified by "H"; packed decimal is denoted by "P"; single precision (short) floating-point is specified by "S"; and double precision (long) floating-point is denoted by "L." All other operations are fullword.

Since storage is only allocated by words in the PDP-10, there are no addressing problems in BLISS-10. Storage can be allocated in several ways on the IBM S/360, as previously mentioned. Therefore, BLISS-360 must provide the capability of specifying the alignment of the variables when storage is allocated. Table IV lists the reserved words which will be used in the allocation of storage and their corresponding boundary alignment.

TABLE IV

LIST OF RESERVED WORDS INDICATING BOUNDARY ALIGNMENT

Type of Variable	Boundary Alignment
GLOBAL	Byte
OWN	Byte
LOCAL	Byte
GLOBALH	Halfword
OWNH	Halfword
LOCALH	Halfword
GLOBALF	Fullword
OWNF	Fullword
LOCALF	Fullword
REGALLOC	Fullword
REGISTER	Fullword
GLOBALD	Doubleword
OWND	Doubleword
LOCALD	Doubleword
FPREGALLOC	Doubleword
FPREGISTER	Doubleword

The IBM S/360 instruction set contains an extensive number of byte manipulation instructions. It may be desirable, however, to have the capability of manipulating characters without resorting to inline machine language coding. This can be provided by incorporating a modified version of the BLISS-10 "pointer parameter" into BLISS-360. The BLISS-360 version of the"pointer parameter," called the Bit/Byte Modifier, will provide the capability of accessing a field consisting of a contiguous number of bits, or bytes. It will have the general format of :

$E_o < E_1 , E_2 , E_3 >$

where E_o is the base address, E_1 indicates the modifier type (0 indicates Byte access; 1 indicates Bit access), E_2 is the starting position, and E_3 is the size of the field. In addition, E_2 and E_3 may be optionally omitted in which case the default values of 0 and 1, respectively, will be assigned.

It should be noted that the format of the Bit/Byte Modifier can only be viewed as a guideline for future implementation because use of the symbols " < " and " > " or parenthesis as delimiters will cause conflicts with the "structure access" and "function call" constructs.

Examples of the Bit/Byte Modifier will illustrate its use. Assume the variables SEND and RECEIVE are aligned on doubleword boundaries, with default structure Vector and size one (doubleword).

```
RECEIVE : = .SEND
RECEIVE : = .SEND <0,0,8>
```

In both cases, the entire value of send is stored into RECEIVE.

```
RECEIVE : = .SEND <0,4,2>
```

In this case, only the value of the field containing the fifth and sixth bytes is stored into RECEIVE.

Finally, the "plit" construct must be incorporated into the grammar.

Thus, although all BLISS-10 constructions do not appear in the present BLISS-360 grammar, sufficient constructs exist to allow compiler development to continue while the grammar is extended as a separate effort.

IV. CONCLUSIONS

A system software production language, called BLISS-360, was designed for the IBM S/360 computer. As a modification of the PDP-10 "implementation language" BLISS-10, BLISS-360 is similar in construction. However, it differs from BLISS-10 in those constructs relevant to the hardware capabilities of the host computer.

The design of BLISS-360 is based upon the production of a compiler using the XPL Compiler Generator System. The complete design of the language, including the future additions described, will provide BLISS-360 with all the capabilities of BLISS-10 and will permit it to take full advantage of the architecture of the IBM S/360. Thus, the BLISS-360 language provides the foundation for future development of a BLISS-360 compiler.

APPENDIX A

BNF DESCRIPTION OF BLISS-360

```
 1  <MODULE>  ::=  <MODULE HEAD> <MODULE END>

 2  <MODULE HEAD>  ::=  <HEADING> <EQE>
 3                   |  <HEADING> <ID CLAUSE>

 4  <HEADING>  ::=  MODULE

 5  <MODULE END>  ::=  ELUDOM

 6  <E>  ::=  <LABELLED EXPR>
 7         |  <UNLABELLED EXPR>

 8  <LABELLED EXPR>  ::=  <LABEL> : <UNLABELLED EXPR>

 9  <UNLABELLED EXPR>  ::=  <BALANCED EXPRESSION>
10                        |  <UNBALANCED EXPRESSION>

11  <BALANCED EXPRESSION>  ::=  <SIMPLE EXPRESSION>
12                            |  <CONTROL EXPRESSION>
13                            |  <TEST TRUE PART> <BAL PART>

14  <UNBALANCED EXPRESSION>  ::=  <IF CLAUSE> <FIN THEN CLAUSE>
15                              |  <TEST TRUE PART> <UNBAL PART>

16  <IF CLAUSE>  ::=  IF <E>

17  <TEST TRUE PART>  ::=  <IF CLAUSE> <THEN CLAUSE>

18  <THEN CLAUSE>  ::=  <THEN> <BALANCED EXPRESSION>

19  <BAL PART>  ::=  ELSE <BALANCED EXPRESSION>

20  <UNBAL PART>  ::=  ELSE <UNBALANCED EXPRESSION>

21  <SIMPLE EXPRESSION>  ::=  <P11>
22                         |  <P11> <RIGHT PART>

23  <RIGHT PART>  ::=  := <E>

24  <P11>  ::=  <P10>
25           |  <P11> XOR <P10>
26           |  <P11> EQV <P10>
```

```
27   <P10>  ::=  <P9>
28            |  <P10> OR <P9>

29   <P9>   ::=  <P8>
30            |  <P9> AND <P8>

31   <P8>   ::=  <P7>
32            |  NOT <P7>

33   <P7>   ::=  <P6>
34            |  <P6> <RELATION> <P6>

35   <P6>   ::=  <P5>
36            |  - <P5>
37            |  <P6> + <P5>
38            |  <P6> - <P5>

39   <P5>   ::=  <P4>
40            |  <P5> * <P4>
41            |  <P5> / <P4>
42            |  <P5> MOD <P4>

43   <P4>   ::=  <P3>
44            |  <P4> | <P3>

45   <P3>   ::=  <P1>
46            |  . <P3>

47   <P1>   ::=  <LITERAL>
48            |  <STRUCTURE ACCESS>
49            |  <FUNCTION CALL>
50            |  <BLOCK>

51   <RELATION>  ::=  EQL
52                  |  NEQ
53                  |  LSS
54                  |  LEQ
55                  |  GTR
56                  |  GEQ

57   <LITERAL>  ::=  <IDENTIFIER>
58                 |  <NUMBER>
59                 |  <STR>

60   <STR>  ::=  <STRING>
61           |  <STRING TYPE> <STRING>
```

```
62   <STRING TYPE>  ::=  ASCII

63   <STRUCTURE ACCESS>  ::=  <STRUCTURE ACCESS HEAD> <E> >

64   <STRUCTURE ACCESS HEAD>  ::=  <IDENTIFIER> <
65                              |  <STRUCTURE ACCESS HEAD> <E> ,

66   <FUNCTION CALL>  ::=  <FUNCTION HEAD> <EPAR>
67                      |  <FUNCTION HEAD> )

68   <FUNCTION HEAD>  ::=  <IDENTIFIER> (
69                      |  <FUNCTION HEAD> <E> ,

70   <BLOCK>  ::=  <BLOCK HEAD> <BLOCK END>
71               |  <BL HEAD4> <BLOCK END>

72   <BLOCK HEAD>  ::=  <BL HEAD1> <DECLARATION>
73                    |  <BL HEAD2> <E>

74   <BL HEAD1>  ::=  <BL HEAD4>
75                  |  <BL HEAD1> <DECLARATION> ;

76   <BL HEAD2>  ::=  <BL HEAD1>
77                  |  <BL HEAD2> <E> ;

78   <BL HEAD4>  ::=  BEGIN
79                  |  (

80   <BLOCK END>  ::=  END
81                  |  )

82   <CONTROL EXPRESSION>  ::=  <LOOP EXPRESSION>
83                           |  <ESCAPE EXPRESSION>
84                           |  <CHOICE EXPRESSION>
85                           |  <COROUTINE EXPRESSION>

86   <LOOP EXPRESSION>  ::=  <WHILE EXPRESSION>
87                        |  <UNTIL EXPRESSION>
88                        |  <DO WHILE EXPRESSION>
89                        |  <DO UNTIL EXPRESSION>
90                        |  <INCREMENT EXPRESSION>
91                        |  <DECREMENT EXPRESSION>

92   <WHILE EXPRESSION>  ::=  <WHILE CLAUSE> <DO CLAUSE>

93   <WHILE CLAUSE>  ::=  WHILE <E>
```

```
 94  <DO CLAUSE>  ::=  DO <E>
 95  <UNTIL EXPRESSION>  ::=  <UNTIL CLAUSE> <DO CLAUSE>
 96  <UNTIL CLAUSE>  ::=  UNTIL <E>
 97  <DO WHILE EXPRESSION>  ::=  <DO CLAUSE> <WHILE CLAUSE>
 98  <DO UNTIL EXPRESSION>  ::=  <DO CLAUSE> <UNTIL CLAUSE>
 99  <INCREMENT EXPRESSION>  ::=  <INCR HEAD> <RIGHT ID PART>
100  <INCR HEAD>  ::=  INCR <IDENTIFIER>
101  <RIGHT ID PART>  ::=  <STEP EXPRESSION> <DO CLAUSE>
102  <STEP EXPRESSION>  ::=  <STEP PART> <ID PART>
103                       |  <STEP PART>
104  <STEP PART>  ::=  <START STEP> <END STEP>
105  <START STEP>  ::=  FROM <E>
106  <END STEP>  ::=  TO <E>
107  <ID PART>  ::=  BY <E>
108  <DECREMENT EXPRESSION>  ::=  <DECR HEAD> <RIGHT ID PART>
109  <DECR HEAD>  ::=  DECR <IDENTIFIER>
110  <ESCAPE EXPRESSION>  ::=  <ENVIRONMENT ESCAPE>
111                          |  <PROCEDURE ESCAPE>
112  <ENVIRONMENT ESCAPE>  ::=  <LOOP ESCAPE>
113                          |  <CONTROL ENVIRONMENT ESCAPE>
114  <LOOP ESCAPE>  ::=  <EXIT>
115                   |  <EXIT EXPRESSION>
116  <EXIT>  ::=  EXITLOOP
117  <EXIT EXPRESSION>  ::=  EXITLOOP <EXIT END>
118  <EXIT END>  ::=  <BRACED E>
119                |  <ESCAPE VALUE>
120                |  <BRACED E> <ESCAPE VALUE>
```

```
121   <BRACED E>  ::=  < <E> >

122   <ESCAPE VALUE>  ::=  <WITH CLAUSE>

123   <WITH CLAUSE>  ::=  WITH <UNLABELLED EXPR>

124   <CONTROL ENVIRONMENT ESCAPE>  ::=  <LABELLED LEAVE>
125                                    |  <LABELLED LEAVE> <ESCAPE VALUE>

126   <LABELLED LEAVE>  ::=  LEAVE <IDENTIFIER>

127   <PROCEDURE ESCAPE>  ::=  <RETURN>
128                          |  <RETURN EXPRESSION>

129   <RETURN>  ::=  RETURN

130   <RETURN EXPRESSION>  ::=  RETURN <ESCAPE VALUE>

131   <CHOICE EXPRESSION>  ::=  <CHOICE HEAD> <CHOICE END>

132   <CHOICE HEAD>  ::=  <LEFT CHOICE> <E> OF

133   <LEFT CHOICE>  ::=  CASE
134                     |  SELECT
135                     |  <LEFT CHOICE> <E> ,

136   <CHOICE END>  ::=  <CASE END>
137                   |  <SELECT END>

138   <CASE END>  ::=  <SET HEAD> TES
139                  |  <SET HEAD> <E> TES

140   <SET HEAD>  ::=  SET
141                  |  <SET HEAD> ;
142                  |  <SET HEAD> <E> ;

143   <SELECT END>  ::=  <NSET HEAD> TESN
144                    |  <NSET HEAD> <NE> <E> TESN

145   <NSET HEAD>  ::=  NSET
146                   |  <NSET HEAD> <NE> <E> ;

147   <NE>  ::=  <XE>
148            |  ALWAYS :
149            |  OTHERWISE :

150   <XE>  ::=  <UNLABELLED EXPR> :
```

```
151   <COROUTINE EXPRESSION>  ::=  <CREATE EXPRESSION>
152                              |  <EXCHJ EXPRESSION>

153   <CREATE EXPRESSION>  ::=  <CREATE HEAD> <CREATE END>

154   <CREATE HEAD>  ::=  <CREATE> <FUNCTION CALL>

155   <CREATE>  ::=  CREATE

156   <CREATE END>  ::=  <LOC CLAUSE> <LENGTH CLAUSE> <FIN THEN CLAUSE>

157   <LOC CLAUSE>  ::=  AT <E>

158   <LENGTH CLAUSE>  ::=  LENGTH <E>

159   <FIN THEN CLAUSE>  ::=  <THEN> <UNLABELLED EXPR>

160   <THEN>  ::=  THEN

161   <EXCHJ EXPRESSION>  ::=  <EXCHANGE HEAD> <EPAR>

162   <EXCHANGE HEAD>  ::=  <EXCHANGE> <BL HEAD4>
163                        |  <EXCHANGE HEAD> <E> ,

164   <EXCHANGE>  ::=  EXCHJ

165   <EPAR>  ::=  <E> <BLOCK END>

166   <DECLARATION>  ::=  <TYPE DECLARATION>
167                     |  <ML DECLARATION>

168   <ML DECLARATION>  ::=  <SINGLE ML DECLARATION>
169                         |  <ALL ML DECLARATION>

170   <ALL ML DECLARATION>  ::=  ALLMACHOP

171   <SINGLE ML DECLARATION>  ::=  <ML HEAD> <ID CLAUSE>

172   <ML HEAD>  ::=  MACHOP
173                |  <ML HEAD> <ID CLAUSE> ,

174   <ID CLAUSE>  ::=  <IDENTIFIER> <EQE>

175   <EQE>  ::=  = <E>
```

```
176  <TYPE DECLARATION>  ::=  <FUNCTION DECLARATION>
177                        |  <STRUCTURE DECLARATION>
178                        |  <ALLOCATION DECLARATION>
179                        |  <BIND DECLARATION>
180                        |  <REGISTER DECLARATION>
181                        |  <LABEL DECLARATION>

182  <LABEL DECLARATION>  ::=  <LABEL HEAD> <IDENTIFIER>

183  <LABEL HEAD>  ::=  LABEL
184                  |  <LABEL HEAD> <IDENTIFIER> ,

185  <STRUCTURE DECLARATION>  ::=  <STRUCTURE HEAD> <EQE>
186                             |  <STRUCTURE HEAD> <STRUCTURE END>

187  <STRUCTURE HEAD>  ::=  <STRUCTURE NAME>
188                      |  <STRUCTURE NAME> <BRAC ID> <IDENTIFIER> >

189  <STRUCTURE NAME>  ::=  STRUCTURE <IDENTIFIER>

190  <BRAC ID>  ::=  <
191               |  <BRAC ID> <IDENTIFIER> ,

192  <STRUCTURE END>  ::=  = <BRACED E> <E>

193  <FUNCTION DECLARATION>  ::=  <TYPE FUNCTION>
194                            |  <FUNCTION TYPE LOC>

195  <TYPE FUNCTION>  ::=  <TYPE FUNCTION HEAD> <EQE>

196  <TYPE FUNCTION HEAD>  ::=  <PARAMLESS FUNCTION>
197                          |  <PARAM FUNCTION> <IDENTIFIER> )

198  <PARAM FUNCTION>  ::=  <PARAMLESS FUNCTION> (
199                       |  <PARAM FUNCTION> <IDENTIFIER> ,

200  <PARAMLESS FUNCTION>  ::=  <FUNCTION TYPE> <IDENTIFIER>

201  <FUNCTION TYPE>  ::=  FUNCTION
202                     |  ROUTINE
203                     |  GLOBALROUTINE

204  <FUNCTION TYPE LOC>  ::=  <TYPE LOC HEAD> <FUNCTION CALL>

205  <TYPE LOC HEAD>  ::=  EXTERNAL
206                     |  FORWARD
207                     |  <TYPE LOC HEAD> <FUNCTION CALL> ,
```

```
208   <ALLOCATION DECLARATION>  ::=  <ALLOC1>

209   <ALLOC1>  ::=  <ALLOC HEADING> <STRUCTURE ALLOC>
210              |   <ALLOC HEADING> <IDENTIFIER>
211              |   <ALLOC HEADING> <STRUCTURE ACCESS>
212              |   <ALLOC0> <IDENTIFIER>
213              |   <ALLOC0> <STRUCTURE ACCESS>

214   <ALLOC0>  ::=  <ALLOC1> :

215   <STRUCTURE ALLOC>  ::=  <IDENTIFIER> <IDENTIFIER>
216                        |  <IDENTIFIER> <STRUCTURE ACCESS>

217   <ALLOC HEADING>  ::=  <ALLOCATE HEAD>
218                      |  <BIND HEAD>

219   <ALLOCATE HEAD>  ::=  <ALLOCATE TYPE>
220                      |  <ALLOC1> ,

221   <ALLOCATE TYPE>  ::=  GLOBAL
222                      |  OWN
223                      |  LOCAL
224                      |  EXTALLOC
225                      |  REGALLOC
226                      |  FPREGALLOC
227                      |  MAP

228   <BIND HEAD>  ::=  <BIND HEADING>
229                  |  <ALLOC2> ,

230   <BIND HEADING>  ::=  BIND

231   <BIND DECLARATION>  ::=  <ALLOC2>

232   <ALLOC2>  ::=  <ALLOC1> <EQE>

233   <REGISTER DECLARATION>  ::=  <REGISTER HEAD> <REGISTER END>

234   <REGISTER HEAD>  ::=  REGISTER
235                      |  FPREGISTER
236                      |  <REGISTER HEAD> <REGISTER END> ,

237   <REGISTER END>  ::=  <ID CLAUSE>
238                     |  <STRUCTURE ACCESS> <EQE>
```

BIBLIOGRAPHY

1. Wulf, W.A., Russell, D.B., and Habermann, A.N., "BLISS: A Language for Systems Programming," Communications of the ACM, v. 14, p. 780-790, December 1971.

2. Richards, M., "BCPL: A Tool for Compiler Writing and System Programming," Proceedings of the AFIPS 1969 Spring Joint Computer Conference, v. 34, p. 557-566.

3. Brender, R.F., BLISS-11 and BLISS-10 Contrasts, Pgh: Carnegie-Mellon University, internal memorandum, 14 January 1972.

4. Knuth, D.E., Fundamental Algorithms-The Art of Computer Programming, v. 1, Addison-Wesley, 1969.

5. Bahler, R.C., Steps Toward a Compiler for BLISS-360, M.S. Thesis, Naval Postgraduate School, Monterey, 1972.

6. PDP-10 Reference Handbook, Digital Equipment Corporation, 1970.

7. International Business Machines Corporation, IBM System/360 Principles of Operation, 8th ed.

8. McKeeman, W.M., Horning, J.J., and Wortman, D.B., A Compiler Generator, Prentice-Hall, 1970.

9. BLISS Reference Manual, Pgh: Carnegie-Mellon University, Computer Science Department, 15 January 1970.

10. Bauer, H.R., and others, Algol W Language Description, Stanford: Stanford University, Computer Science Department, September 1969.

11. Malcolm, M.A., PL360 (Revised) A Programming Language for the IBM 360, Stanford: Stanford University, Computer Science Department, May 1971.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Asst Professor G. A. Kildall, Code 53Kd Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
4.	Maj. Eugenia Mary Zavoyski, USMCR 137 South River Street Plains, Pennsylvania 18705	1

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Naval Postgraduate School Monterey, California 93940	UNCLASSIFIED
	2b. GROUP

3. REPORT TITLE

Preliminary Design of the Systems Implementation Language BLISS-360

4. DESCRIPTIVE NOTES (Type of report and, inclusive dates)

Master's Thesis; June 1972

5. AUTHOR(S) (First name, middle initial, last name)

Eugenia Mary Zavoyski

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1972	49	11

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY

13. ABSTRACT

The preliminary specification of a systems implementation language, called BLISS-360, is given. BLISS-360 is intended to be used for generation of software for the IBM S/360 computer, and is patterned after BLISS-10 which was originally designed for the PDP-10. The Backus-Naur Form description of BLISS-360 is given in a form acceptable to the Mixed-Strategy Precedence parser of the XPL Compiler Generator System. Thus, the XPL System can be used to aid future developmental efforts leading to a completed compiler for BLISS-360.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

A-31408

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
BLISS						
Programming Language						
Systems Implementation Language						
XPL Compiler Generator System						
Expression language						

Thesis
Z248
c.1
135462
Zavoyski
Preliminary design of the systems implementation language BLISS-360.